Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za individualni projekat

Student: Sladaković Milica, SV 18/2020

Predmet: Paralelno programiranje

Tema projektnog zadatka: Detekcija ivica unutar slike

SADRŽAJ

1. Uvod i analiza problema

Digitalna obrada slike predstavlja složen i zahtjevan postupak, za čiju implementaciju se u realnom vremenu najčešće koriste specijalizovani sklopovi sa digitalnim signal procesorima, multiprocesorski i distribuirani sistemi i druge specijalizovane arhitekture za obradu slike. Neke od osnovnih operacija za obradu slike su: operacije zasnovane na histogramu, filtriranje u prostornom i frekvencijskom domenu, izdvajanje ivica, poboljšanje i restauracija slike, te geometrijske operacije.

Tema ovog projekta je paralelizacija, možda najčešće korištene operacije nad slikom, detekcije ivica. U nastavku će biti pokazana paralelizacija dva algoritma za izdvajanje ivica – algoritam sa Prewitt operatorom prvog reda i algoritam sa pretragom okoline svakog piksela. Za implementaciju programa, potrebno je da sve slike budu BMP formata sa RGB prostorom boja, jer se za parsiranje slike koristi biblioteka `EasyBMP`. Radi jednostavnosti, obrada će se vršiti nad crno-bijelim verzijama ulaznih slika. Sve izlazne slike će biti skalirane, bez sive komponente (crne, sa bijelim ivicama).

Takođe, analiziraće se performanse paralelizacije u C++ programskom jeziku, gdje se paralelizacija ostvaruje korištenjem raspoređivača zadataka iz Intel-ove biblioteke Thread Building Blocks. Mjerenja će biti vršena na Intel i7-1065G7 četvorojezgarnom procesoru (sa osam logičkih jezgara) i Windows 10 Education operativnom sistemu.

2. Koncept rješenja

U osnovi digitalne obrade slike nalazi se operacija dvodimenzionalne diskretne konvolucije. Konvolucija je matematička operacija nad dvije funkcije i proizvodi treću funkciju, koja opisuje kako se ponašanje prve funkcije mijenja zbog druge funkcije. Ova operacija se zasniva na kernelu (kvadratnoj matrici malih dimenzija) koji kliže preko svih podmatrica ulazne slike (podmatrice su dimenzija kernela) i obavlja aritmetičke operacije nad elementima podmatrice, te rezultate smješta u izlaznu matricu. Elementi matrice slike sadrže podatke o vrijednostima boja pojedinih tačaka (piksela) slike.

U nastavku su objašnjena dva algoritma nad kojima će se vršiti paralelizacija.

2.1. Algoritam sa Prewitt operatorom

U obradi slike, Prewitt operator je kernel, odnosno filter, koji aproksimira gradijent. Gradijent predstavlja usmjerenu promjenu intenziteteta boja na slici. Ovaj filter vrši množenje korespodentnih elemenata dvije matrice i sumira rezultat koji se smješta u izlaznu matricu. Sljedeća formula opisuje filtriranje ulazne slike koja se nalazi u ulaznoj matrici X, filterom jezgra F dimenzije $f \times f$, i smještanje rezultata u izlaznu matricu Y:

$$Y[x,y] = \sum_{n=0}^{f} \sum_{m=0}^{f} X\left[x - \frac{f}{2} + m, y - \frac{f}{2} + n\right] * F[m,n]$$

Ulazna slika se filtrira u dva pravca – horizontalnom i vertikalnom. Dakle, u jednom pravcu se primjenjuje operator, a u drugom pravcu invertovan operator. Time se dobijaju dvije vektorske komponente, horizontalna i vertikalna. Konačan rezultat filtriranja podmatrice je vrijednost vektorskog zbira horizontalne i vertikalne komponente filtriranja. Dati su primjeri operatora (G_x i G_y respektivno za horizontalnu i vertikalnu obradu):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \qquad \overrightarrow{|G|} = |\overrightarrow{G_x} + \overrightarrow{G_y}|$$

2.2. Algoritam sa pretragom okoline svakog piksela

Okolina piksela predstavlja susjedne tačke piksela: lijevo, desno, iznad, ispod i dijagonalno na sve strane. Ovakva obrada slike zahtijeva najprije uklanjanje sive boje skaliranjem vrijednosti ulazne matrice na vrijednosti 0 i 1 (u odnosu na prag koji iznosi 128). Ovo rezultuje crno-bijelom slikom bez izdvojenih ivica. Potom se pretražuje okolina svakog piksela na sljedeći način:

- $P(i,j) = 1$, ako u okolini tačke postoji tačka sa vrednošću 1
- $P(i,j) = 0$, ako u okolini tačke ne postoji tačka sa vrednošću 1
- $O(i,j) = 0$, ako u okolini tačke postoji tačka sa vrednošću 0
- $O(i,j) = 1$, ako u okolini tačke ne postoji tačka sa vrednošću 0

Vrijednost koja se upisuje u izlaznu matricu je apsolutna vrijednost razlike $P(i,j) - O(i,j)$ vraćena u opseg 0 – 255.

3. Programsko rješenje

Za implementaciju programa, potrebno je da sve slike budu BMP formata sa RGB prostorom boja, jer se za parsiranje slike koristi biblioteka `EasyBMP`. Radi jednostavnosti, obrada će se vršiti nad crno-bijelim verzijama ulaznih slika. Sve izlazne slike će biti skalirane, bez sive komponente (crne, sa bijelim ivicama).

Cilj implementacije je da rješenje bude korektno i da ne zavisi od veličine slike, veličine filtera i veličine okruženja piksela. U nastavku su date implementacije najvažnijih funkcija.

Parametri komandne linije su nazivi fajlova (ulazni fajl, izlazni fajlovi za serijsku i paralelnu verziju oba algoritma). Takođe, u zaglavlju programa su definisane konstante `FILTER_SIZE` (dimenzije filtera), `filterHor` i `filterVer` (vrijednosti horizontalne i vertikalne komponente filtera), `NEIGHBOURHOOD_SIZE` (veličina okruženja koje se posmatra) i `CUTOFF` (granični slučaj dimenzije matrice do kog se kreiranju novi paralelni zadaci).

Tokom rada, mjeri se trajanje svake od obrada, a na kraju i verifikacija rezultata, odnosno poklapanje rezultata dobijenih serijskom i paralelnom obradom.

3.1. Implementacija algoritma sa Prewitt operatorom

Data je implementacija funkcije za filtriranje. Povratna vrijednost funckije je skaliran rezultat operatora (0 ili 255, uklanjanjem sive boje).

```
/**
* @brief Prewitt operator on input image submatrix around pixel
*
* @param inBuffer buffer of input image
* @param x horizontal coordinate of pixel
* @param y vertical coordinate of pixel
*
* @return scaled value of operator result
```

```
*/
int filter(int* inBuffer, int x, int y) {

  int save = FILTER_SIZE / 2;
  int G = 0, Gx = 0, Gy = 0, raw;
  for (int n = 0; n < FILTER_SIZE; n++) {
          for (int m = 0; m < FILTER_SIZE; m++) {
                  raw = inBuffer[(x - save + m) + (y - save + n) * totalWidth];
                  Gx += raw * filterHor[m + n * FILTER_SIZE];
                  Gy += raw * filterVer[m + n * FILTER_SIZE];
          }
  }
  G = sqrt(Gx * Gx + Gy * Gy);
  return 255 * scale(G);
}
```

Serijska verzija funkcije za detekciju ivica pomoću Prewitt operatora:

```
/**
* @brief Serial version of edge detection algorithm implementation using Prewitt
operator
*
* @param inBuffer buffer of input image
* @param outBuffer buffer of output image
* @param width image width
* @param height image height
* @param col current column of input image
* @param row current row of input image
*/
void filter_serial_prewitt(int* inBuffer, int* outBuffer, int width, int height,
int col, int row)
{
  int lowerX, lowerY, upperX, upperY;
  int skip = FILTER_SIZE / 2 + 1;

  lowerX = (col < skip) ? skip : col;
  lowerY = (row < skip) ? skip : row;
  upperX = (col + width > totalWidth - skip) ? totalWidth - skip    :  col + width;
  upperY = (row + height > totalHeight - skip) ? totalHeight - skip : row + height;

  for (int x = lowerX; x < upperX; x++) {
          for (int y = lowerY; y < upperY; y++) {
                  outBuffer[x + y * totalWidth] = filter(inBuffer, x, y);
          }
  }
}
```

Na početku funkcije se vrši prilagođavanje polaznih i krajnjih vrijednosti reda i kolone, kako se ne bi izašlo iz opsega početne matrice i pristupilo pogrešnoj memorijskoj lokaciji. Zatim se prolazi kroz cijelu matricu i filtriranje podmatrica se smješta u odgovarajuće elemente izlazne matrice.

U ovom postupku je uočeno zgodno mjesto za paralelizaciju. Prolasci kroz matricu se mogu paralelizovati tako što se cijela matrica podijeli na manje cjeline, nad kojima se vrši obrada. Dodatno, svaki dio može da se dijeli do određene granice (definisani CUTOFF), te se tako postiže veća količina paralelizacije. Za svaki dio početne matrice zadužen je jedan zadatak raspoređivača. Ovaj „podijeli i zavladaj“ algoritam, zajedno sa većim brojem zadataka, doprinosi značajnom ubrzanju programa, o čemu će biti pričano u narednom poglvlju.

Paralelna verzija funkcije za detekciju ivica pomoću Prewitt operatora:

```
/**
* @brief Parallel version of edge detection algorithm implementation using Prewitt
operator
*
* @param inBuffer buffer of input image
* @param outBuffer buffer of output image
* @param width image width
* @param height image height
* @param col current column of input image
* @param row current row of input image
*/
void filter_parallel_prewitt(int* inBuffer, int* outBuffer, int width, int height,
int col, int row)
{
  if (min(height, width) <= CUTOFF) {
          filter_serial_prewitt(inBuffer, outBuffer, width, height, col, row);
  }

  else {
          task_group t;
          int taskHeight = height / 2;
          int restHeight = height - taskHeight;
          int taskWidth = width / 2;
          int restWidth = width - taskWidth;
          t.run([&]    {filter_parallel_prewitt(inBuffer,    outBuffer,    taskWidth,
taskHeight, col, row); });
          t.run([&]    {filter_parallel_prewitt(inBuffer,    outBuffer,    restWidth,
taskHeight, col+taskWidth, row); });
          t.run([&]    {filter_parallel_prewitt(inBuffer,    outBuffer,    taskWidth,
restHeight, col, row+taskHeight); });
          t.run([&]    {filter_parallel_prewitt(inBuffer,    outBuffer,    restWidth,
restHeight, col+taskWidth, row+taskHeight); });
          t.wait();
  }
}
```

Prije podjele matrice na podmatrice, potrebno je osigurati da neki od redova ili kolona neće biti preskočeni usljed ostatka pri dijeljenju. Zbog toga se dimenzije krajnjih ivičnih podmatrica računaju kao razlika odgovarajućih dimenzija početne matrice i sume iste dimenzije svih ostalih podmatrica u jednom pravcu.

3.2. Implementacija algoritma sa pretragom okoline svakog piksela

Data je funkcija za pretragu okoline piksela:

```
/**
* @brief Neighbourhood check around input image pixel
*
* @param inBuffer buffer of input image
* @param x horizontal coordinate of pixel
* @param y vertical coordinate of pixel
*
* @return scaled value of neighbourhood result
*/
int checkNeighbours(int* outBuffer, int x, int y) {
  int P = 0, O = 1;
  int value;
```

```
  int skip = NEIGHBOURHOOD_SIZE / 2 + 1;
  for (int i = 0; i < NEIGHBOURHOOD_SIZE; i++) {
          for (int j = 0; j < NEIGHBOURHOOD_SIZE; j++) {
                  value = scale(outBuffer[(x - i + skip) + (y - j + skip) *
totalWidth]);
                  if (value == 1) P = 1;
                  else O = 0;
          }
  }
  return 255 * abs(P - O);
}
```

Serijska verzija funkcije za detekciju ivica pretragom okoline piksela:

```
/**
* @brief Serial version of edge detection algorithm
*
* @param inBuffer buffer of input image
* @param outBuffer buffer of output image
* @param width image width
* @param height image height
* @param col current column of input image
* @param row current row of input image
*/
void filter_serial_edge_detection(int* inBuffer, int* outBuffer, int width, int
height,
int col, int row)
{
  int lowerX, lowerY, upperX, upperY;
  int skip = NEIGHBOURHOOD_SIZE / 2 + 1;

  lowerX = (col < skip) ? skip : col;
  lowerY = (row < skip) ? skip : row;
  upperX = (col + width > totalWidth - skip) ? totalWidth - skip    : col + width;
  upperY = (row + height > totalHeight - skip) ? totalHeight - skip : row + height;


  for (int x = lowerX; x < upperX; x++) {
          for (int y = lowerY; y < upperY; y++) {
                  outBuffer[x + y * totalWidth] = checkNeighbours(inBuffer, x, y);
          }
  }
}
```

Kao i kod funkcije sa Prewitt operatorom, na početku se vrši prilagođavanje polaznih i krajnjih vrijednosti reda i kolone, a rezultat se smješta u odgovarajuće elemente izlazne matrice.

Ova funkcija se od Prewitt operatora razlikuje samo po kernel matrici kojom filtrira sliku, te su i mjesto za paralelizaciju i sam postupak paralelizacije identičan.

Paralelna verzija funkcije za detekciju ivica pretragom okoline piksela:

```
/**
* @brief Parallel version of edge detection algorithm
*
* @param inBuffer buffer of input image
* @param outBuffer buffer of output image
* @param width image width
* @param height image height
* @param col current column of input image
```

```
 * @param row current row of input image
 */
void filter_parallel_prewitt(int* inBuffer, int* outBuffer, int width, int height,
int col, int row)
{
  if (min(height, width) <= CUTOFF) {
          filter_serial_edge_detection(inBuffer, outBuffer, width, height, col, row);
  }
  else {
          task_group t;
          int taskHeight = height / 2;
          int restHeight = height - taskHeight;
          int taskWidth = width / 2;
          int restWidth = width - taskWidth;
          t.run([&]  {filter_parallel_edge_detection(inBuffer,  outBuffer,  taskWidth,
taskHeight, col, row); });
          t.run([&]  {filter_parallel_edge_detection(inBuffer,  outBuffer,  restWidth,
taskHeight, col+taskWidth, row); });
          t.run([&]  {filter_parallel_edge_detection(inBuffer,  outBuffer,  taskWidth,
restHeight, col, row+taskHeight); });
          t.run([&]  {filter_parallel_edge_detection(inBuffer,  outBuffer,  restWidth,
restHeight, col+taskWidth, row+taskHeight); });
          t.wait();
  }
}
```

4. Ispitivanje rezultata

4.1. Karakteristike računara

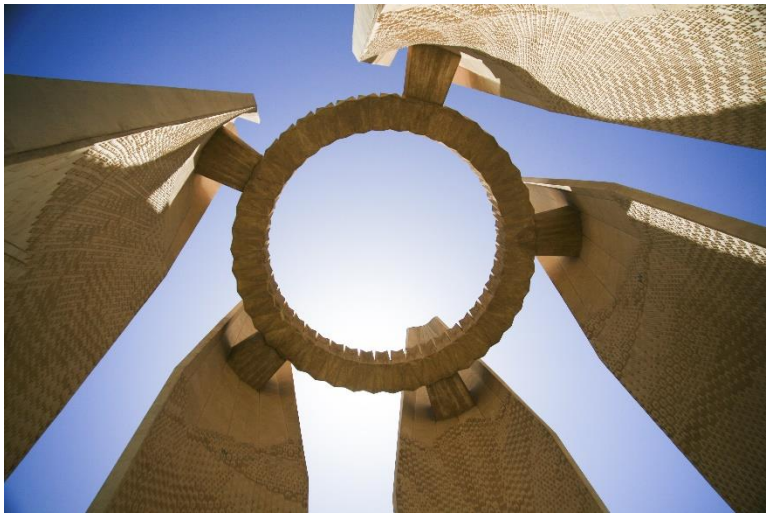
Mjerenja su vršena na Intel i7-1065G7 četvorojezgarnom procesoru (sa osam logičkih jezgara i baznom frekvencijom od 1.50 GHz) i Windows 10 Education operativnom sistemu.

4.2. Testni slučajevi

Program je testiran nad četiri različite ulazne slike, a parametri koji su mijenjani su: vrijednost filtera (`filterHor` i `filterVer`) i njegova veličina (`FILTER_SIZE`), veličina okruženja piksela koja se analizira (`NEIGHBOURHOOD_SIZE`), te krajnja granica za paralelizaciju (`CUTOFF`).

Među ulaznim slikama, koje su raznih dimenzija, nalazi se animirana slika, slika sa čistom pozadinom (obje slike nemaju mnogo detalja, te je pronalazak ivica računski nezahtijevan), uspravna slika sa detaljima i slika sa mnoštvom detalja.

U nastavku slijede rezultati dobijeni obradom navedenih slika sa određenim vrijednostima parametara.



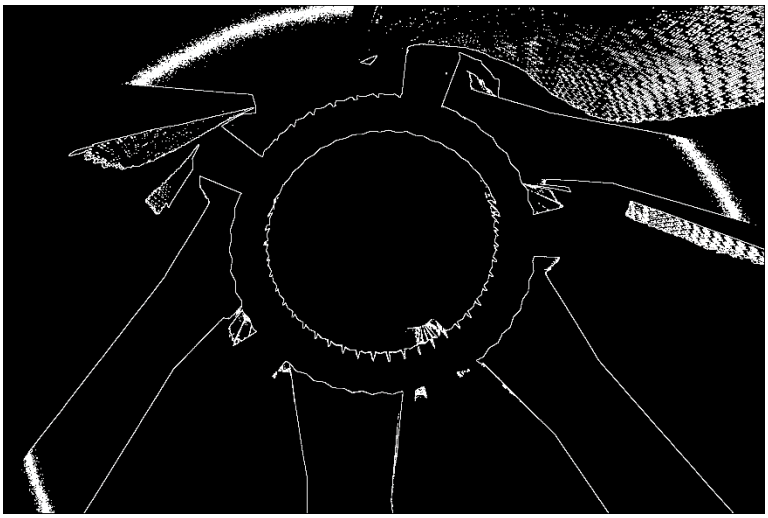
Primjer 1. [architecture]

Veličina slike: 3888×2592

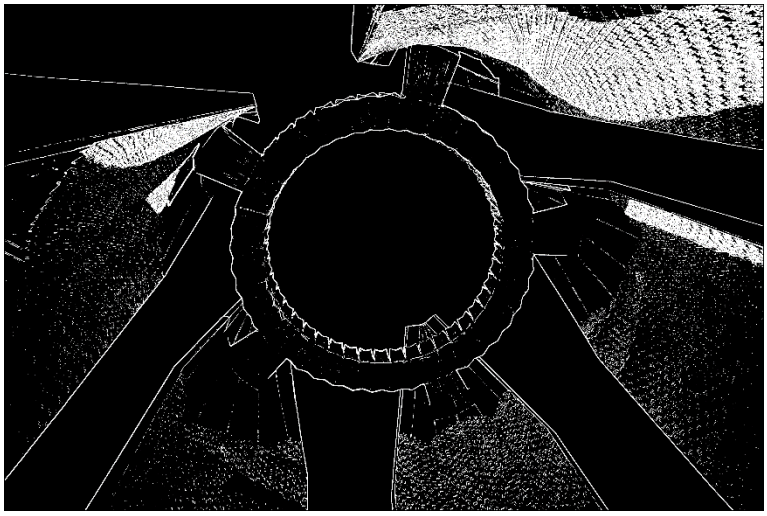
$$G_x = \begin{bmatrix} -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \end{bmatrix}$$

Veličina okruženja: 5

Slika nema mnogo detalja, te je primjetno da iako malo manje detaljna, pretraga okoline svakog piksela radi podjednako dobar posao kao i Prewitt operator. Tokom testiranja, zapaženo je da povećanjem dimenzija operatora i okruženja dolazi do prikaza suviše detalja van ivica na izlaznoj fotografiji.



Primjer 1. [architecture] – pretraga okoline piksela



Primjer 1. [architecture] – Prewitt operator

Primjer 2. [kamenko]

Primjer 2. [kamenko] – pretraga okoline piksela

Primjer 2. [kamenko] – Prewitt operator

Veličina slike: 736×1279

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

Veličina okruženja: 9

Slika je animirana, te je pronalazak ivica računski nezahtijevan, a uz to, i dimenzije same slike su male. Tokom testiranja, primjećeno je da sve veličine operatora rade jednako dobar posao, dok manje veličine okruženja bolje detektuju ivice (sa povećanjem ovog parametra, ivice postaju zadebljanije i manje precizne).

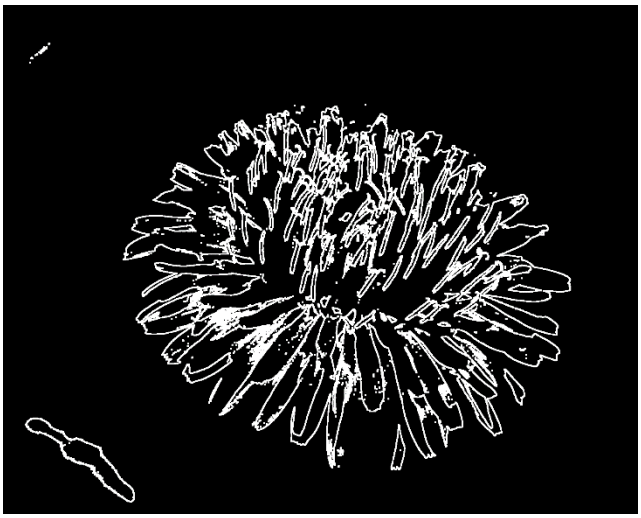
Primjer 3. [dandelion]

Veličina slike: 1765 × 1413

$$G_x = \begin{bmatrix} -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \\ -1 & -1 & -1 & 0 & +1 & +1 & +1 \end{bmatrix}$$

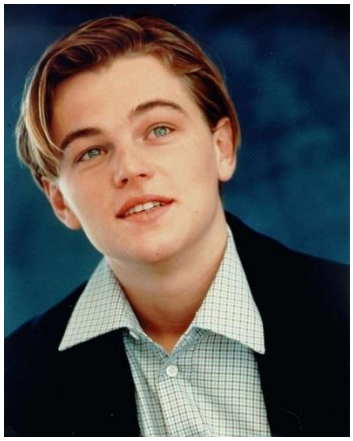
Veličina okruženja: 5

Slika ima mnoštvo detalja i, ponovo, Prewitt operator se pokazao detaljnijim, ali je i drugi metod pronašao suštinu. Tokom testiranja, zapaženo je da povećanjem veličine okruženja dolazi do suviše zadebljanih ivica.



Primjer 3. [dandelion] – pretraga okoline piksela

Primjer 3. [dandelion] – Prewitt operator



Primjer 4. [dicaprio]

Primjer 4. [dicaprio] – pretraga okoline piksela

Primjer 2. [dicaprio] – Prewitt operator

Veličina slike: 959 × 1200

$$G_x = \begin{bmatrix} -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \end{bmatrix}$$

Veličina okruženja: 8

Tokom testiranja primjećeno da je da povećanjem dimenzija operatora, izlazna slika postaje previše detaljna i gubi suštinu.

Navedeni primjeri predstavljaju najbolje rezultate dobijene testiranjem obrade sa raznim vrijednostima parametara.

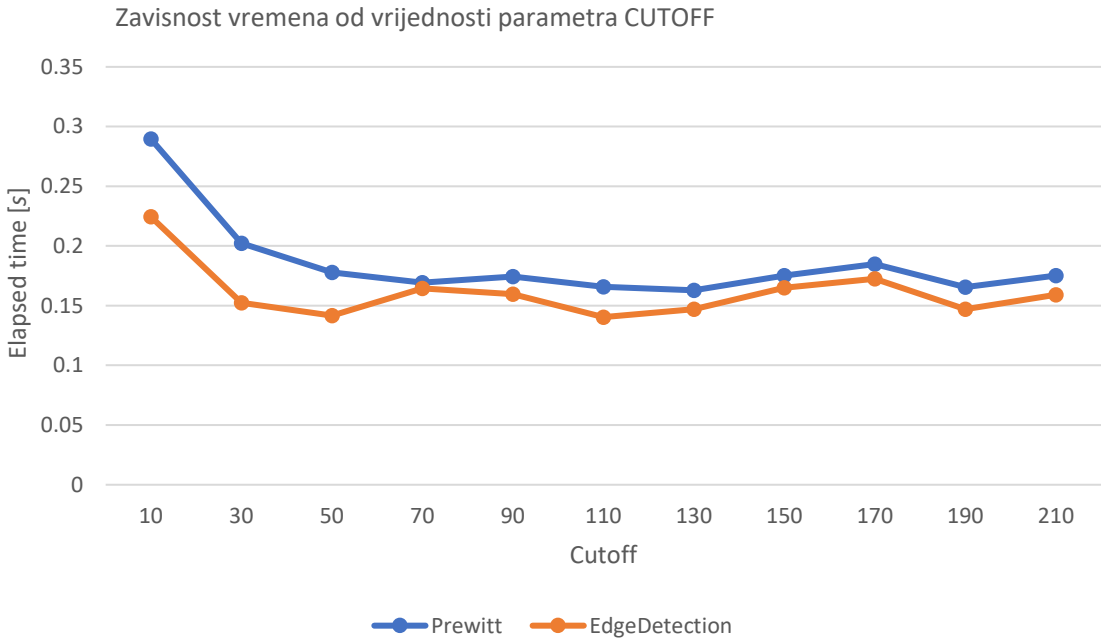
4.3. Rezultati

Vrijednost `CUTOFF` parametra određena je tako što je više puta pokrenut algoritam obrade nad raznim vrijednostima ovog parametra.

Na donjem grafiku prikazana je zavisnost vremena od vrijednosti ovog parametra. Mjerenje se vršilo na slici većih dimenzija (3888 × 2592). Ispostavilo se da je za trodimenzionalne podmatrice u oba algoritma optimalna vrijednost CUTOFF parametra oko 70 (prvi algoritam) i 110 (drugi algoritam), a sve iz intervala 50-150 za prvi i 30-50, te 100-130 za drugi algoritam daje solidne rezultate. Za sva dalja mjerenja se uzima vrijednost parametra odsijecanja 100 za Prewitt operator i 50 za pretragu okoline.

U tabeli su date vrijednosti mjerenja.

Cutoff	10	30	50	70	90	110	130	150	170	190	210
Prewitt	0.289	0.202	0.177	0.169	0.174	0.165	0.162	0.175	0.184	0.165	0.175
EdgeDetection	0.224	0.152	0.141	0.164	0.159	0.140	0.146	0.165	0.172	0.146	0.159

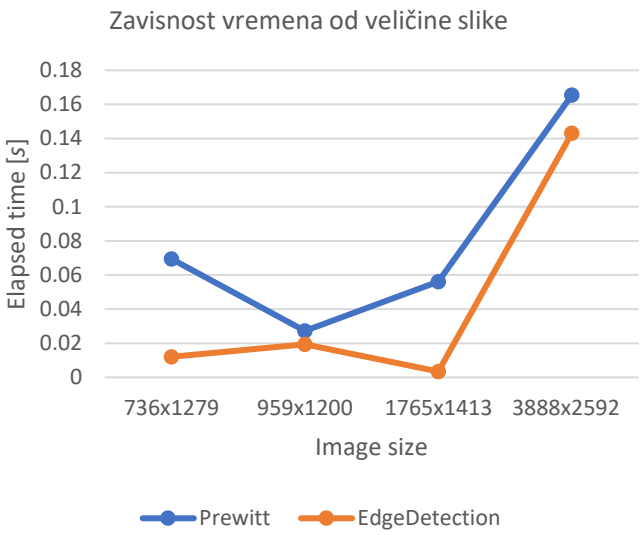


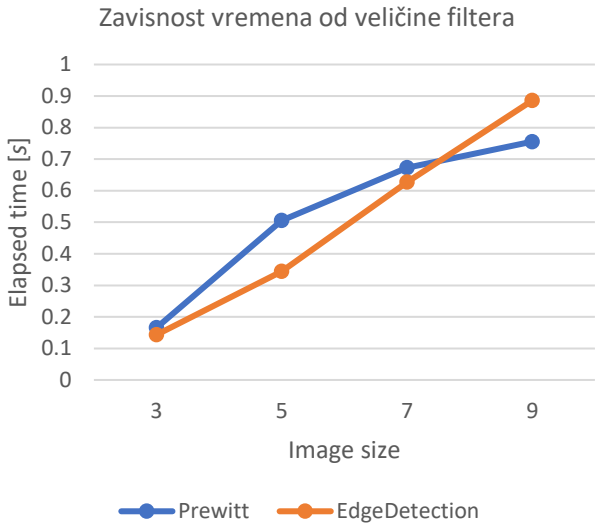
Na lijevom grafiku prikazana je zavisnost vremena izvršenja od veličine slike. Sva naredna mjerenja će, radi što bolje analize, biti izvršena na najvećoj dimenziji slike.

Na desnom grafiku prikazana je zavisnost vremena izvršenja od veličine kernela. Sasvim je intuitivno da povećanjem kernela i trajanje obrade raste, te bi se, ukoliko bi se procijenilo da veći kernel značajno povećava krajnji rezultat, mogla paralelizovati i sama obrada podmatrice.

U tabeli su dati rezultati mjerenja.

	Image size				Kernel size			
	736x1279	959x1200	1765x1413	3888x2592	3	5	7	9
Prewitt	0.069443	0.027247	0.056215	0.16548	0.16548	0.5051	0.672409	0.755486
EdgeDetection	0.012036	0.019433	0.003457	0.143177	0.143177	0.343901	0.627089	0.8859



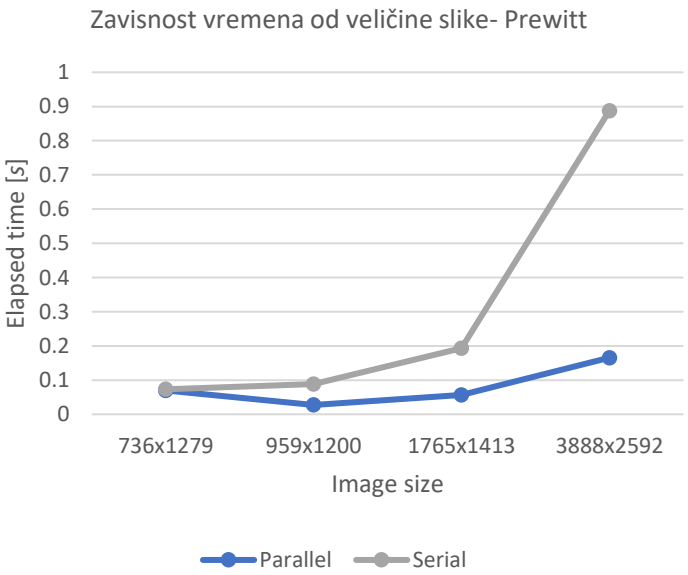


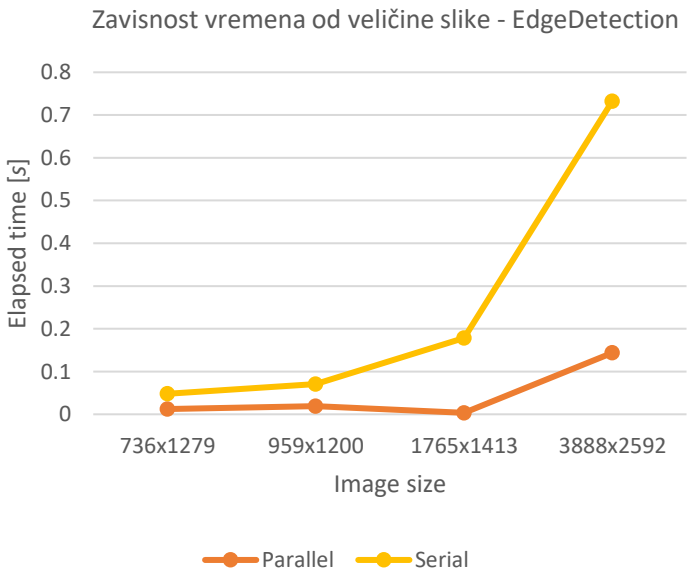
Na graficima se vidi da povećanje dimenzija Prewitt operatora podsjeća na logaritamsku funkciju, dok povećanje dimenzija okruženja koje se analizira za drugi algoritam izaziva eksponencijalno ponašanje zavisnosti vremena. Takođe, vidi se da je zavisnost vremena od povećanja dimenzija približno eksponencijalno zavisno.

Na kraju, važno je uporediti trajanje obrade serijskim i paralelizovanim algoritmima. Iz narednih grafika, jasno se vidi da je uspostavljeno ubrzanje u vrijednosti između 4 i 5.

U tabeli su dati rezultati mjerenja.

	Image size	736x1279	959x1200	1765x1413	3888x2592
Prewitt	Parallel	0.069448	0.027247	0.056213	0.16548
	Serial	0.073158	0.08805	0.192808	0.887779
EdgeDetection	Parallel	0.012368	0.019342	0.0034568	0.143777
	Serial	0.047963	0.070991	0.178671	0.732602





5. Analiza rezultata

Svi dobijeni rezultati određeni su usrednjavanjem rezultata više mjerenja. Većina mjerenja je imala približan ishod, dok su neka mjerenja značajno odudarala od ostatka rezultata.

Iz gore navedenih rezultata, može se zaključiti da ubrzanje paralelizacije značajno zavisi od veličine slike, pa i od veličine kernela. Ukoliko bi došlo do toga da je potrebno obraditi sliku ogromnih dimenzija, svakako da ni trenutno rješenje u nekim situacijama ne bi bilo zadovoljavajuće. Ukoliko bi, uz to, bilo potrebno koristiti kernel velikih dimenzija, dalja paralelizacija se čini neophodnom. Dakle, ovakvo rješenje daje još prostora za paralelizaciju problema.

Pored svega navedenog, značajno ubrzanje (i do 10 puta) je uočeno prilikom paralelizacije sa više paralelnih zadataka. Ovakva podjela problema se dobro pokazala pri malim vrijednostima parametra `CUTOFF`.

Obzirom na četvorojezgarni procesor, baš ovakvo ubrzanje je i očekivano, te se paralelizacija pokazala uspješnom i korisnom, a zaključak je da bi za dalji razvoj bilo potrebno prepoznati nova kritična mjesta u programu i implementirati njihovu paralelizaciju.